Open Source Hardware Design and Hardware Reverse Engineering: a Security Analysis

Johanna Baehr, Alexander Hepp, Michaela Brunner, Maja Malenko and Georg Sigl

August 4, 2022

Open Source Hardware Design and Hardware Reverse Engineering: A Security Analysis

Johanna Baehr*, Alexander Hepp*, Michaela Brunner*, Maja Malenko‡ and Georg Sigl*†
*Chair of Security in Information Technology, TUM School of Computation, Information and Technology
Technical University of Munich, Germany
{johanna.baehr, alexander.hepp, michaela.brunner, sigl}@tum.de
†Fraunhofer Institute for Applied and Integrated Security (AISEC), Germany
‡Hensoldt Cyber GmbH, Germany
maja.malenko@hensoldt.net

***Abstract*—Major industry-led initiatives such as RISC-V and OpenTitan strive for verified, customizable and standardized products, based on a combination of Open Source Hardware (OSHW) and custom intellectual property (IP), to be used in safety and security-critical systems. The protection of these products against reverse-engineering-based threats such as IP Theft and IP Piracy, Hardware Trojan (HT) insertion, and physical attacks is of equal importance as for closed source designs. OSHW generates novel threats to the security of a design and the protection of IP. This paper discusses to what extent OSHW reduces the difficulty of attacking a product. An analysis of the reverse engineering process shows that OSHW lowers the effort to retrieve broad knowledge about a product and decreases the success of related countermeasures. In a case study on a RISC-V core and an AES design, the red team uses knowledge about OSHW to circumvent logic locking protection and successfully identify the functionality and the used locking key. The paper concludes with an outlook on the secure protection of OSHW.**

***Index Terms*—reverse engineering, open source hardware, IP protection, IC trust, IP obfuscation, hardware security**

I. Introduction

The past years have seen a rise in the availability of Open Source Hardware (OSHW) designs. Both the RISC-V Community and the commercial partnership behind the OpenTitan Project [1] have driven the availability of complex and verified OSHW processor designs. Groups like the OpenHW Group [2] create high-quality OSHW designs with industrial quality verification, with 9 RISC-V cores currently under development. Many other companies are also dedicated to producing fully open source processors. OSHW implementations of cryptographic algorithms are commonly produced or even required for competitions on cryptographic algorithms [3] and are often also freely available on the internet. Websites such as OpenCores have provided implementations ranging from arithmetic cores to interfaces and entire processors since 1999 [4].

This development is of interest in particular for processor designs. In the past, many processor designs were created by one company and used by others as Third-Party Intellectual Property (3PIP) black or gray boxes. One prominent example, ARM Ltd, developed the original advanced RISC machine (ARM) architecture, and licenses the architecture or its implementations to different companies [5]. Other companies prefer to develop their own architectures in-house, or buy the license for an existing architecture and develop only the implementation in-house. For both scenarios, the core can only be verified by the licensing company or in-house, and cannot be easily verified by an external party.

New development in OSHW allows for a more complete verification and an increased trust in the design. Since every party can verify the implementation, and the community actively supports this effort, security through obscurity is replaced by security through verification. A prominent example is OpenTitan [1], an open-source project to design a silicon root-of-trust. The project aims for transparency, trustworthiness, and security, by open-sourcing as much of the design process as possible. In particular for companies or research institutes unable to develop their own processor design, using OSHW is a low-cost and verifiable method to build a chip. Furthermore, as the source code is open source rather than a black box, the design is easily customizable, and own Intellectual Property (IP) can easily be added to the design.

The effect of using OSHW in design for security, however, should be studied. When knowledge of a hardware design is made public, this information is made public to attackers, too. In Integrated Circuit (IC) trust and security, this has a direct effect on the ability to reverse engineer the design [6]. This can lead to three attack scenarios:

- IP theft or piracy
- Reverse Engineering (RE) to insert Hardware Trojans (HTs)
- RE to identify attack vectors

Each of these scenarios can occur during different steps in the life of a chip (see Figure 1). IP Theft or IP Piracy occurs once the design leaves the company, either for verification, fabrication or once it becomes available on the market and can be cloned by an end user. RE to identify attack vectors is done by an end user unless the attacker is actively working with a foundry. Finally, HT insertion can occur either via 3PIP

This work was partly funded by the German Ministry of Education and Research in the project RESEC under Grant No.: 16KIS1009.

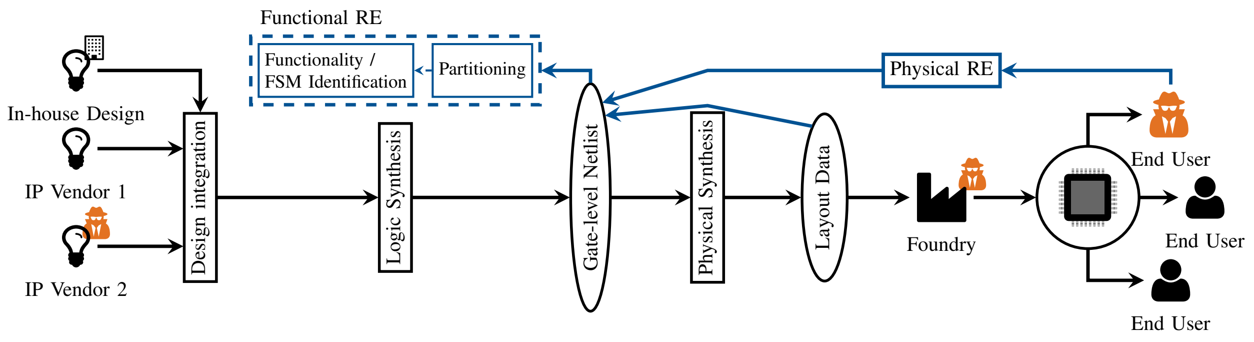


Figure 1: Life of a chip, attack procedures (blue) and attack scenario entry points (orange)

at the beginning of the design flow, or during fabrication by the foundry.

The additional attacker knowledge provided by the OSHW design elements has a different impact on each of the three attack scenarios. This additional knowledge also affects countermeasures against reverse engineering, which are simpler to break if the entire design or part of the design is open source.

The effect of using OSHW, and consequently the effect on the success of RE and the corresponding attacks depends on the extent of the used OSHW. Similar to software design, in silicon hardware development, OSHW can be used as a building block for proprietary designs, or the design is OSHW in its entirety.

This work will analyze the effect of previous design knowledge on each of the three attack scenarios. In section II, the attack scenarios affected by OSHW will be discussed. Section III describes the impact of OSHW on the level of difficulty to perform netlist abstraction during RE. Section IV discusses the effect on RE countermeasures. Section V presents a red team vs blue team based case study of the RE of components of a RISC-V design and an AES design. The functionality of an unknown but OSHW design is recovered, and the logic obfuscation broken using a SAT-Attack. Finally, viable countermeasures are discussed in section VI.

II. Risks and Attack Scenarios

A. *IP Theft and IP Piracy*

If solely OSHW is used for the development of a chip, IP piracy is not a significant risk. If the chip is reverse engineered to clone the design, the only effort stolen is the effort to lay out and verify the design for the specific technology. For a foundry, this may be economically beneficial, however, for an end user, the effort to reverse engineer the chip from the ground up is too high to be viable. Generally, neither overbuilding, nor patent infringement or business secret leakage must be protected against.

If the OSHW is complemented by proprietary in-house designs such as cryptographic accelerators, interfaces, or other implementations, the risk significantly increases. As explained in section III and shown in section V, the use of OSHW reduces the difficulty to identify design components and highlights the proprietary parts, consequently making IP piracy of these components much simpler.

B. *Hardware Trojan Insertion*

The threat of HT insertion in 3PIP is minimized, if not eliminated, by the use of OSHW designs, as the design can be verified easily, not only by the designer, but by anyone using the design. However, the threat of HT insertion during fabrication at the foundry increases significantly. As with IP Piracy, if the entire design, or components of the design, are open source, the effort to reverse engineer the design is greatly reduced, and thus, it becomes much easier to identify suitable insertion points for HTs. Previous work shows that it is possible to insert a variety of HTs into a RISC-V design without high overhead [7] If the design consists of OSHW as well as proprietary hardware, the foundry may decide to forgo identification of the additional hardware, and only use the open source components of the design to insert a HT.

C. *Identification of Attack Vectors*

This attack, most commonly carried out by the end user, seeks to identify vulnerable points within the chip to carry out subsequent attacks, such as side channel, fault or probing attacks. Generally, the effort to reverse engineer the design from the chip as an end user is relatively large, so that it only becomes viable for a chip in significant attacks scenarios, for example by state actors. However, if the attacker is able to work together with a foundry, and thus has access to layout data, this kind of attack becomes viable for any chip. Again, if the design, or part of the design used for the chip is open source, the effort to reverse engineer and identify suitable attack vectors is significantly reduced.

Additionally, an adversary can leverage its knowledge about OSHW to search for vulnerabilities in the OSHW sources instead of reverse engineering the hardware. This knowledge can be acquired through manufacturer's public announcements, datasheets or by (possibly superficial) RE. For example, an attacker could inspect the source code of a RISC-V design and find vulnerabilities such as [8], a severe bug in the case study's RISC-V design that allows to trigger a machine-level Denial of Service (DoS) from user-level software.

In summary, the attack scenarios of interest when considering OSHW are (1) IP Theft, when OSHW is mixed with proprietary hardware, (2) the insertion of HTs by the foundry, and (3) the identification of design weaknesses to be exploited by subsequent hardware attacks.

III. Impact on the Reverse Engineering Workflow

To understand the impact of open source knowledge of the design on each of these attack scenarios, we must understand how RE is impacted. It is clear that the most probable and prominent attack scenarios on OSHW occur when the attacker has access to the netlist. Furthermore, physical RE, as described in [9], i.e., the processes to retrieve the netlist from the physical chip, is not significantly improved by knowledge of the design. Thus, we will focus on functional RE of the netlist, which was gained either through extraction from layout data from the foundry, or through physical RE, see Figure 1. Functional RE, also known as netlist abstraction, generally consists of two parts: (1) the partitioning of the entire design into smaller submodules, and (2) the identification of the functionality of these submodules. Some submodules might represent or contain a Finite State Machine (FSM). An FSM is determined by first identifying state flip-flops, and then recovering the FSM [10].

A. Netlist Partitioning

Partitioning seeks to group the logic gates of a netlist into functional submodules. Solutions can be divided into data-path-identification-based and graph-based methods. In [11], the data path is identified by grouping similar gates into words, and finding the connections between these. Partitioning is then done by cutting out modules between connected groups of words. This idea was later improved by combining control signal based data path identification, PCA based methods and similarity scores [12]. Graph-based methods leverage the idea that functional submodules will create structural clusters within the netlist (i.e., form follows function). Several approaches have been proposed: early methods use graph clustering algorithms to identify possible submodules [13], later circuit embeddings were used to generate hierarchical clusters [14]. More recently, graph neural networks (GNNs) have been used to partition the netlist [15].

However, none of these methods can partition the netlist into perfect and functional submodules. Instead, in most cases errors occur, i.e., logic gates are assigned to the wrong submodule. Furthermore, each method has parameters, which must be chosen correctly to achieve quality results. For data-path based methods, such parameters are common word sizes, or metrics to calculate similarity between gates. For graph-based methods, parameters distinctive to the clustering method are required, which will often control the number, size, and density of the resulting clusters. For machine learning based approaches, parameters are required for feature extraction, as well as for the machine learning hyperparameters.

In an OSHW, the partitioning is at least partially known. This provides feedback to tune the parameters and to verify the resulting partitioning, increasing the quality of the results. Even when only some components of the design are open source, it becomes simple to separate this part from unknown or proprietary components, greatly reducing the effort to partition the rest of the design.

B. Identification of Functionality

The second step concerns the identification of the functionality of the submodules. In general, this is done by comparison to something known, called the golden model. The golden model can be described, among other possibilities, as a template based description or a boolean description of the functionality, an Register Transfer Level (RTL) implementation or gate-level netlist, or as a structural interpretation.

To test whether a design matches a golden model within a golden model library (i.e., a large set of known designs, compiled of smaller and larger design building blocks), the functionality can be identified using either formal methods (more exact) [11], or fuzzy methods (less exact) [15], [16]. Fuzzy methods are used when the design contains errors, either from physical reverse engineering, or due to partitioning, as errors within the submodules mean that formal methods can no longer be used [16]. However, a perfect partitioning into submodules allows for the use of formal methods if the submodules are not design-specific. A perfect partitioning thus facilitates a more exact identification of the functionality.

When the design is unknown (closed source), submodules will be matched against standard building blocks and known commercial IP (e.g., arithmetic circuits, interfaces, etc). This results in many smaller identified submodules and some unidentified submodules, which must then be puzzled together into a larger functional description of the design. However, if the design is open source, it becomes trivial to add the entire design as well as all submodules to the golden model library, and thus a match is guaranteed. In the best case, partitioning of the design is not needed, as the entire design can be matched.

If the design is customized, or if only part of the design is open source, proprietary hardware and customisations can be easily partitioned out (section III-A). The focus could then lie on identifying the functionality of unknown, proprietary hardware, because the additional OSHW can be identified easily and efficiently. Proprietary hardware will be especially valuable, because it is the most custom and specialized part of the design, making it attractive for attacks, like IP theft.

C. FSM Identification

A common way to reconstruct an FSM is to first identify which flip-flops contain state information [17]. This information and the knowledge of the reset state are then used to extract the FSM [10], [18]. The first step is the more difficult, as state flip-flops must be identified out of all flip-flops in the design or submodule, and no perfect method yet exists [17], [19]. One approach is to identify state flip-flops as being less similar in their fan-in behavior compared to other flip-flops, but this comparison requires a number of parameters, and results vary in quality [17], [19].

The reconstruction of an FSM can be significantly simplified when using OSHW. If the correct number of state flip-flops and their fan-in behavior are known, state flip-flop identification methods based on [17] can perform considerably better, because knowledge about the targeted state flip-flops can support the parameter choice. Additionally, matching-based attacks are usually not applicable for FSMs, because of their high variability [18], [20]. However, if an open source design and a perfect partition are used, the perfect matching library could also allow the identification of the respective FSMs.

IV. Impact on common Reverse Engineering Countermeasures

RE countermeasures can be divided into three categories: split-manufacturing, logic locking and IC camouflaging [21]. Both IC camouflaging and split-manufacturing are layout level methods to thwart the extraction of the complete gate-level netlist, in the first case by the end user, and in the second case by the foundry. OSHW might have few or no impact on these countermeasures, but might increase the weaknesses and high associated costs which exist for many of the proposed implementations [21], [22]. However, in the context of OSHW, we focus on attacks that are based on functional instead of physical RE, so we assume an available gate-level netlist, see section III. Therefore, we discuss the third countermeasure, logic locking, in more detail and show why this method is not adequate to protect OSHW.

Logic locking targets the prevention of IP counterfeiting, IP theft, and HT insertion during the manufacturing process in the foundry [23], [24]. It obfuscates the design by corrupting the output if a wrong secret locking key is applied [25], [26].

There exist several attack methods to overcome logic locking. In [21], attack methods are categorized into four groups: oracle-less, oracle-guided, sequential oracle-guided, and t-probed oracle. Usually, the extraction of the complete, correct locking key is targeted, but there are application scenarios, where an approximate locking key or no locking key is sufficient. For a successful HT insertion, only the location of some specific control wires might be necessary, independent of the remaining, locked circuit [7], [21]. Consequently, the general identification of estimated functionalities of circuit parts might be sufficient.

In an OSHW scenario, all oracles-less threat models become oracle-guided threat models, as the oracle is always provided by the OSHW design. Then, one dominant requirement for a successful attack is information about an unlocked version of the obfuscated design, either in form of an oracle for a SAT-based attack [27], [28] or in form of known and labeled component library entries for a matching-based attack [16]. If logic locking is used to protect OSHW, one can conclude that it has less effect than if it is used for commercial IP. The open source IP serves as information about the unlocked version for the obfuscated design. Consequently, SAT-based attacks will be enabled for products that are not yet available on the market, and matching-based attacks are significantly improved due to a perfect unlocked library component. This will be further explored in a case study in section V.

Table I: Design Metrics and Hierarchy.

	Red Team			Blue Team		
	#PIs	#POs	#Gates	#PIs	#POs	#Gates
ex_stage	586	744	46,656	1,042	939	36,663
– alu 🔒(590)	135	65	2,933	725	65	3,844
– branch_unit	269	264	1,538	207	261	965
– lsu	385	463	16,987	442	1,118	19,705
– mult	142	69	24,297	144	69	12,424
– divider	138	69	3,839	143	69	2,600
– multplier	141	68	20,332	143	69	9,943
key_expand	129	256	7,526	271	256	5,739
– rcon	1	16	81	2	16	70
– S-Box(×8) 🔒(141)	8	8	646	149	8	734

V. Case Study

We use a red team vs blue team approach to illustrate the effect of OSHW on the ability to reverse engineer a design. The blue team seeks to securely fabricate an OSHW-based design, by introducing logic locking into parts of the design. The red team seeks to insert a HT in this design, by identifying the correct functionality of the blue team implementation and breaking the obfuscation. Extracting the correct key and breaking the obfuscation is, as mentioned above, not explicitly necessary for HT insertion, but does allow for easier insertion of larger HTs.

To achieve this goal, the red team first investigates the open source design to identify suitable parameter choices for RE, and to eventually break the obfuscation of the design. Both teams use the same RTL code, but synthesize the designs with different synthesis options and tools, different cell libraries and different optimizations, to create the netlist.

In the first experiment, we show that using the same base design, synthesized using different synthesis tools and technologies, good choices for partitioning parameters correlate. We also show that this is not the case for different designs. We focus on graph-based clustering to partition the designs, as these methods can achieve very good results, but are also very dependent on good parameter choices [13]. The red team then use this information to find an ideal parameter choice to partition the blue team design.

In a second experiment, the red team identifies the functionality of each partition. Using the ideal partitioning from the first experiment, the red team matches each partition of the blue team design against a library of designs, including the red team implementation of the design, to identify the most closely matching module, as done in [16].

Both experiments are done for two scenarios. The first scenario focuses on the functionality identification of part of a RISC-V CVA6 (Ariane) Core [29]. Both teams synthesize the execution stage (`ex_stage`) and the blue team locks specific parts of the design using random logic locking [25]. The second scenario focuses on the key expand module (`key_expand`) of an AES core. In this scenario, the goal of the red team goes beyond functional identification. Exact knowledge of the open source design is used to first partition the design with 100% accuracy. The identification of the

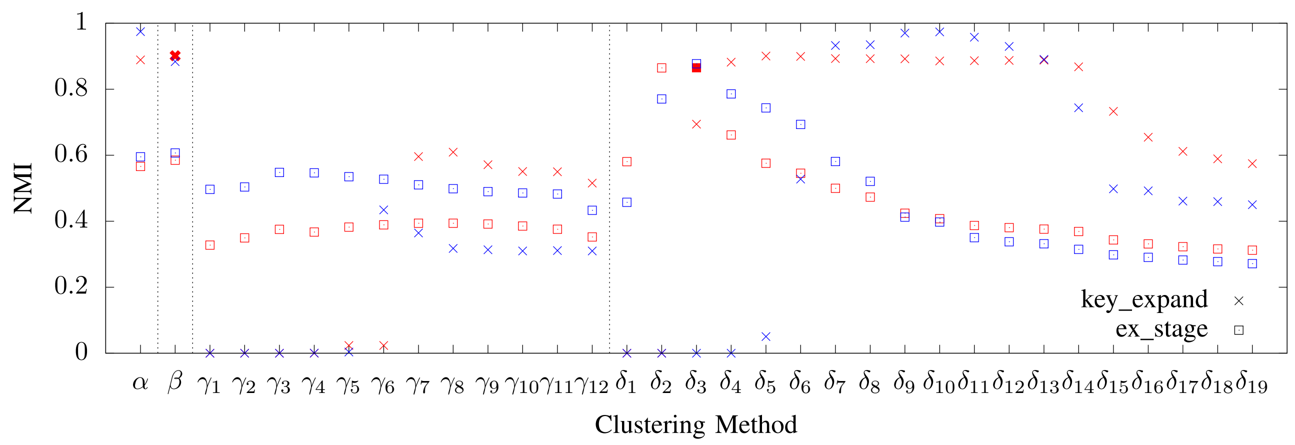


Figure 2: Correlation of the effect of clustering method and parameter choice on the partition quality (NMI) for the red team and blue team partitions, for four clustering methods (α, β, γ, δ), and several parameter choices (indices) on a non-linear scale. The optimum parameters for both designs, as derived from the OSHW by the red team, are highlighted in bold.

correct golden model of a locked S-Box reveals the original functionality. In an additional experiment, the red team seeks to break the logic locking key of the S-box. The perfect functionality identification is a prerequisite to perform a successful SAT-attack on the locked S-box to find the correct key.

Both the `ex_stage` and the `key_expand` include several submodules. The module hierarchy and design metrics are shown in Table I. It shows the number of primary inputs (#PIs), the number of primary outputs (#POs) and the number of logic gates (#Gates) for each implementation and module. Submodules locked by the blue team are marked with a padlock, the number of respective key input bits is given in parentheses. Note that the number of gates differ between the red and blue team implementations due to different synthesis options, tools, cell libraries, and optimizations. Differences in synthesis options can also result in merged or removed unused inputs and outputs, which accounts for the different #PIs and #POs. Furthermore, the #PIs vary greatly due to the addition of the locking key.

A. Correlation of Method and Parameter Choices for Partitioning

One of the main weaknesses of graph-based partitioning methods is that there are a wide variety of methods and parameters to choose from. It is difficult to choose which method to use, if no ground truth is available and no knowledge from a similar design guides the choice of the clustering method and its parameters. Without knowing the ground truth, each method and parameter choice is as valid as every other. While results are generally good, they are often not optimal. It may be possible to combine the results of different methods, but this will also not lead to an optimal method and parameter choice.

However, we can show that for similar designs, the partitioning quality correlates for similar parameters. A good choice of method and parameter can thus be used for similar designs, but does not necessarily translate across designs. We use a set of clustering methods to partition the designs, and assess quality with respect to the ground truth.

To measure the quality of a partitioning, we use the normalized mutual information score (NMI). This is calculated using the estimated entropy of the ground truth of the partitions $H(T)$, the estimated entropy of the cluster-based partitions $H(C)$, and the mutual information between both $I(T;C)$ such that

$$\text{NMI}(T,C) = \frac{2 \times I(T;C)}{[H(T) + H(C)]}$$

A NMI of 0 corresponds to no match between the partitions, and a NMI of 1 corresponds to a perfect match. To measure the quality of a single partition, we use the $f1$ score, as the NMI is not well-defined for single partitions. It takes into account the number of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN), and is calculated as the harmonic mean of precision (P) and recall (R),

$$\text{P} = \frac{TP}{TP+FP}, \quad \text{R} = \frac{TP}{TP+FN},$$

such that

$$f1 = \frac{2(P \times R)}{P+R}.$$

Figure 2 shows the partition quality results for both designs and both the blue and red team implementations. We evaluate four clustering methods (α, β, γ, δ), of which two require parameters (γ, δ). The parameters choices are not linear, but are ordered. For the `key_expand` module, several methods result in partitions with a NMI of 0. Most commonly this is caused when the clustering method returns only one partition that contains every gate, while the ground truth consists of several partitions.

In a real-world scenario, an attacker can never calculate the NMI score for the blue team design, as the attacker does not know the ground truth of the partitions of the blue team implementation. During the attack, the red team only knows the parameters for their implementation, and from these

Table II: Partitioning and Functional Identification Results

	Metric	Red Team	Blue Team	Matching Rank
ex_stage (δ_3)	NMI	0.86	0.88	2
– alu 🔒	$f1$	0.78	1.0	2
– branch_unit	$f1$	0.36	0.38	3
– lsu	$f1$	0.98	0.98	1
– mult				
– divider	$f1$	0.96	0.83	13
– multiplier	$f1$	1.0	0.97	14
key_expand (β)	NMI	0.90	0.88	1
– rcon	$f1$	0.52	0.96	90
– S-Box (x8) 🔒	$f1$	0.96	0.95	1

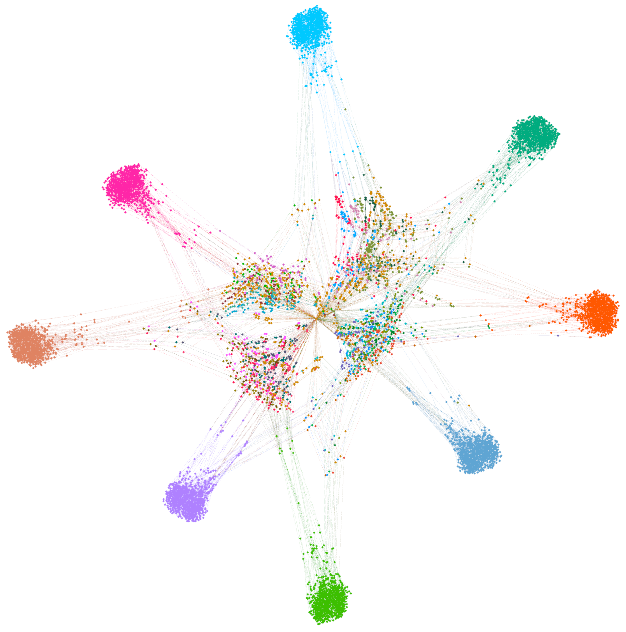
parameters estimates the best parameters for the blue team implementation. For this purpose, the best NMI scores for both red team implementations are highlighted in bold. The red team chooses the corresponding method to attack the blue team implementation.

We can gain two main insights. First, when focusing on the ex_stage design, the partition quality correlates for the red and blue team implementations. A good red team result generally would result in a good result for the blue team implementation. The same applies to the key_expand, although for method δ, the results of the blue team implementation lag behind the results of the red team implementation. Furthermore, choosing the highlighted best method for the red team implementations for both designs results in a good result when attacking the blue team implementation. Thus, we can use knowledge of the OSHW implementation to choose a suitable clustering-based partitioning method and the corresponding parameters. It is not always possible to find the best method, however the results are good enough that the next step can be performed.

The second main insight is that there is no global best method; the results differ for the two different designs. The best choice in method and parameter set for the key_expand is different from the choice for ex_stage and vice versa. Thus, the red team chose method β for the key_expand and method δ_3 for the ex_stage.

The results when choosing these methods for both implementations and their submodule are summarized in Table II. The result for the entire design is calculated using the NMI, while each single submodule is compared using the f1 score against the ground truth. Figure 3 and Figure 4 provide a visual representation of the partitioning of both the red and blue team implementations. Each node corresponds to a netlist standard cell.

For the key_expand (Figure 3), the eight tightly clustered partitions on the outside represent the S-Boxes, while the logic locking key inputs are highlighted in red near the center of Figure 3b. The remaining gates in the center are the rcon module and glue logic. As can be seen in Table II, the rcon was not well identified in the red team implementation, however, in the blue team implementation it could be identified. This can be visually verified as well, in the red team implementation, all gates at the center are badly partitioned, while for the blue team implementation, the rcon consists


(a) Red Team netlist graph for key_expand

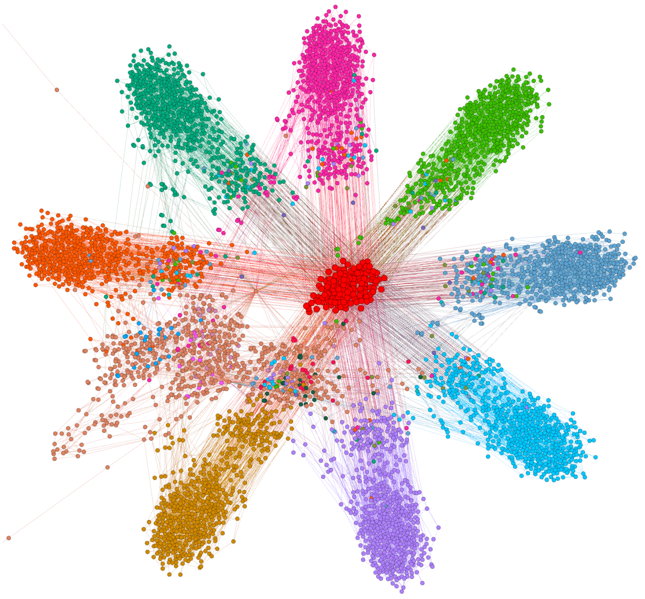

(b) Blue Team netlist graph for key_expand

Figure 3: Comparison of red team and blue team netlists of the key_expand design. Nodes and edges represent cells and wires. Eight clusters on the outside each represent a S-Box. Glue logic and the rcon module are connected to each S-Box and are clustered near the center of each graph. In (b), the logic locking key inputs are shown in red in the center.

of the brown cells between two S-boxes near the bottom left. For both implementations, the S-Boxes are clearly partitioned.

Figure 4 shows the visualization of the ex_stage. The multiplier is on the left side in blue, the lsu is on the right in brown, purple represents the divider and incorporated adder logic and finally the alu is located in the top left side in light green. In the blue team implementation, Figure 4b, the logic locking key gates are also highlighted in red in the `alu`. Table II shows that, for the `ex_stage`, the partitioning results were good for the red team implementation, however, the `alu` and `branch_unit` were partitioned together, resulting in less than ideal $f1$ scores. For the blue team implementation, the `alu` could be partitioned perfectly, this is most likely due to the larger number of gates, due to the added locking mechanism. The good partitioning of the `divider` did not fully transfer to the blue team implementation, since the `branch_unit` was now included in this partition. However, even here, all results are sufficient for functional identification of the modules.

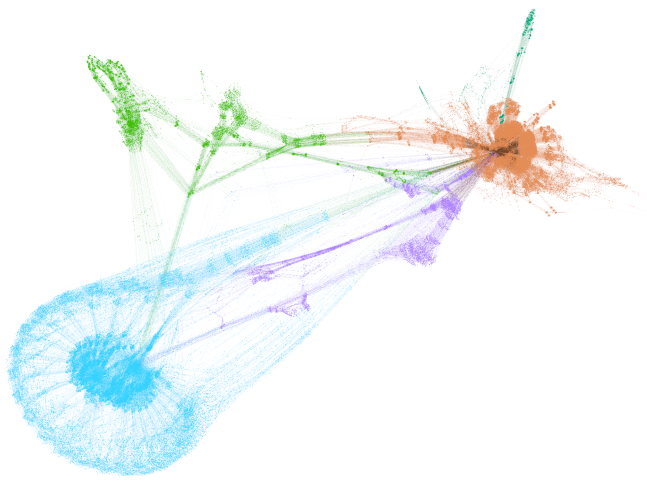
For both designs, the results for the blue team implementation are at least as good, if not better, than the results of the red team implementation. Without previous knowledge regarding parameter and method choice, gained from analyzing the OSHW design, the effort to correctly partition the blue team implementation by the red team would have been significantly higher, as every method is equally viable, but many produce significantly worse results.

B. Functionality Identification

In the second step, the red team seeks to verify whether they can identify the correct golden model for a locked design, to identify to correct functionality. A similar method to [16] was used, where the partitioned (and, if applicable, locked) blue team implementation is the unknown design, and the golden model library consists of approx. 620 other modules of similar sizes. Matching is done based on structural features of the modules. To conform to the open source scenario, the red team implementation was included in the golden model library. The machine learning method outputs a list of decreasingly matching library modules for each partitioned module. Table II shows the rank of the correct module among the list. A rank of 1 is the optimum result, as the closest match is the correct match. A matching rank of 15 implies that 15 other designs were considered a closer match. Even though different synthesis tools, cell libraries and optimizations were used, and the imperfect partitioned modules were used, the golden model could always be identified within the top 15 matches, except for the `rcon` submodule. However, this module contains very few gates, so that the structural features used to describe the module were too few to characterize and match it. Furthermore, the matching ranks of both the `divider` and the `multiplier` were imperfect. However, the highly ranked matches were other multipliers and dividers of different sizes. This occurs because multipliers and dividers are both structurally very similar, changes in size commonly only occur due to different input sizes. It is trivial to manually check which multiplier and divider is the correct one according to the input number and size.

As discussed in section III-B, the best case scenario occurs when the entire design can be matched, as this means that partitioning is not required to completely identify the functionality of the design. This was possible for both the `ex_stage` and `key_expand`. For the attack scenario of HT insertion, both designs are considered broken after this step, as the functionality of each module and submodule is identified. As discussed in section IV, even without recovering the locking key, insertion points for HTs can be found. However, as mentioned, for the insertion of larger and more complex HTs, breaking the locking key can be beneficial.

In a closed source attack scenario, in the best case, no similar design is available for matching in the golden model library, and so this step only identifies similar, but not identical functionality. This greatly increases the security of the design, as the IP is not completely identified, and thus the insertion of HTs becomes significantly more difficult.


(a) Red Team netlist graph for `ex_stage`

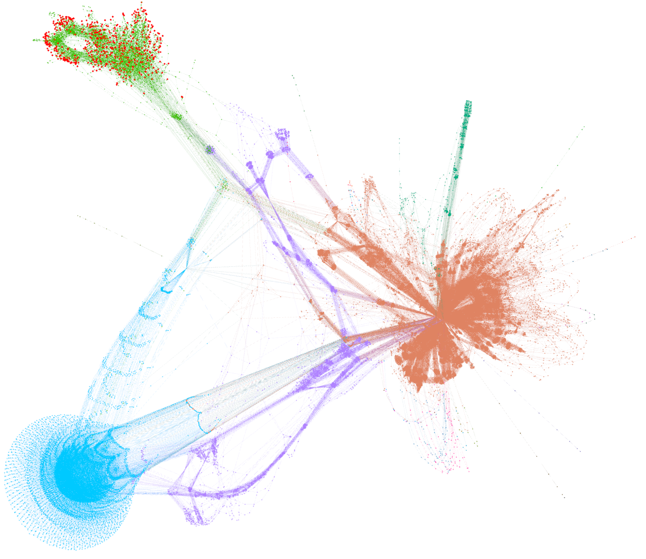

(b) Blue Team netlist graph for `ex_stage`

Figure 4: Comparison of red team and blue team implementation of the `ex_stage` design. The `multiplier` is on the left side in blue, the `lsu` is on the right in brown, purple represents the `divider` and incorporated adder logic, and the `alu` is located in the top left side in light green. In (b) the logic locking key gates are highlighted in red in the `alu`.

C. *Breaking the obfuscated `key_expand`*

In this final experiment, the red team recovers the logic locking key of one of the designs. First, the partitioning of the blue team implementation of the S-box was manually verified, using information from the red team implementation. In particular the locking key inputs were not partitioned to be part of the extracted `S-box`, so these, and the path between the `S-Box` and the inputs were added. Then, the red team implementation, identified to be a matching golden model during the previous experiment, was used as an oracle for a SAT-Attack. It was possible to successfully recover all 141 locking keys of the extracted S-box using the original SAT-based attack [27] on the `key_expand`.

VI. Countermeasures against Netlist Abstraction with Open Source Hardware

A. *Logic Locking*

As discussed in section IV, logic locking based methods which are only secure in an oracles-less threat model are broken in an OSHW scenario and oracle-based attacks are easy to carry out. However, the success of the attack in the above described case study highly depends on practical, real-life considerations. For example, the attack in its current implementation requires the correct mapping of PIs/POs between the original and the locked netlist. Additional constraints on the naming of the signals and the format of the netlist (i.e., simplified Bench format) involve manual efforts, which might be impractical for large circuits. Similarly, we also evaluated RANE [30], an open source CAD-based tool, which performs both oracle-guided and oracle-less attacks on combinational and sequential logic locking. Even though it is a promising tool, with fewer limitations on the format of the circuits and the technology library, RANE still did not always manage to unlock our designs successfully. However, these practical difficulties do not rehabilitate the security of logic locking, as they can be overcome with ease.

Furthermore, SAT-based attacks might become computationally infeasible when unlocking large designs (e.g., full cores) with big key input sizes. In those scenarios, the assumption of a powerful oracle is often deemed to be unrealistic. However, in the context of OSHW, the availability of the golden model (i.e., the powerful oracle), and the ability to partition the design into smaller submodules, makes RE much easier. When done on partitioned and correctly identified modules, the unlocking becomes feasible again. As such, logic locking based countermeasures alone do not provide a realistic method to secure designs against RE and subsequent HT insertion in an OSHW use case.

B. *Customization*

As previously discussed, customized components and additional proprietary hardware will be more easily identifiable when used with OSHW. However, if the open source design is strongly customized, it may protect against simplified RE. In particular when data word sizes, number and functionality of the submodules and the FSMs are changed, the simplified RE process may be hindered. The necessary amount and best type of customization is a subject for future work. Furthermore, this also influences the success of logic locking. If, for example, modified OSHW is used, it might not be applicable as oracle for SAT-based or SAT-related attacks.

C. *Insertion of Dummy Wires*

Correct partitioning is important for the identification of the functionality of the netlist, and thus required to carry out attacks on the design. Hence, methods that increase the errors created during partitioning will also create a more secure design. In the area of IC camouflaging, the idea of using dummy wires and cells has been proposed to make the extraction of the correct netlist more difficult [21]. A similar proposal, where additional wires are added between submodules, which are functionally never reached, may encumber graph-based partitioning methods [31]. The structure and connectivity of the gates will no longer reflect on the functional affiliation of the gates, while the correct functionality is maintained. This obfuscation may be broken by logic optimization, more research into such a method against netlist partitioning is required.

D. *Post-Silicon Testing and Hardware Trojan Detection*

While the pre-silicon verification of IP becomes significantly easier with the use of OSHW, the prevention of HT insertion during fabrication is made more difficult. To counteract this trade-off, more importance could be placed on post-silicon HT detection. Detection methods can be classified into destructive and non-destructive methods [32]. However, methods which require additions or modifications to the design, for example for fingerprinting or Design for Test (DfT) methods, may be circumvented by the foundry more easily in an OSHW scenario.

Test-based approaches generally attempt to verify correct functionality of the chip post fabrication by, if applicable, attempting to activate the HT trigger, and monitoring any abnormal activity [32]. Using OSHW may support this effort, as the required test vectors can be created, verified, and used by the entire community. However, the test vectors are then also known to an attacker. Furthermore, while test-based methods show high potential in identifying many HT, they are not able to identify all types of HT [33].

Side-channel based techniques seek to verify the functionality of the chip compared to a golden circuit by comparing side-channel based characteristics [32]. However, since true golden circuits are not usually available, simulated traces are often used. These have not been shown to successfully identify HT in real-life scenarios. Furthermore, even when a golden chip is available, side-channel abnormalities can also occur due to other factors, and can be difficult to measure precisely enough to successfully identify a HT [32]. Thus, depending on the size, location, complexity and functionality of the HT, non-destructive methods cannot be guaranteed to identify HT.

Invasive methods are significantly more costly, but also significantly more accurate and successful, allowing for a complete verification of the chip. Larger and more complex HT may be identified through optical analysis of the top few metal layers of the IC [32], however, for a complete verification of the design, a full optical reverse engineering of the design is necessary [34].

In general, the specific attack scenario and cost-benefit trade-off should always be considered before choosing the appropriate countermeasures [6]. We believe that a combination of the above discussed countermeasures can be used to successfully ensure and verify that a OSHW-based design is unmodified during fabrication.

VII. Conclusion

While OSHW is low cost, verifiable and customizable, it does allow for easier reverse engineering of the functionality of the design. This allows for easier attacks against the security of the IC. The strength of this effect depends on the attack scenario, as well as on how much of the design is open source. However, mixing open source and proprietary hardware is also not the solution, as OSHW greatly simplifies the functional RE of the proprietary hardware.

We show that logic locking is insufficient for the OSHW scenario. In particular, RE countermeasures that are only deemed secure without an oracle can be easily broken in the OSHW scenario. To protect a device with OSHW against RE-based attacks, novel logic locking solutions need to be developed that strive to make the circuit indistinguishable in its form and function from a random boolean function [35].

However, OSHW also improves security by simplifying RE for beneficial use cases. Simplified RE allows one to find HTs using manual inspection or using a RE-based HT-detection methods with better accuracy. Although we show that OSHW is facing new security challenges, we strongly believe that it is the correct way towards achieving provable security against HT insertion and other hardware attacks.

References

[1] lowRISC contributors. "OpenTitan homepage." (2020), https : / / opentitan.org/.

[2] OpenHW Group. "OpenHW Group," https://www.openhwgroup.org/.

[3] D. J. Bernstein. "CAESAR call for submissions." (2014), https :// competitions.cr.yp.to/caesar-call.html.

[4] OpenCores.org. "OpenCores Homepage." (2021), https://opencores.org/.

[5] ARM Ltd. "ARM Homepage." (2021), https://www.arm.com/products/silicon-ip-cpu.

[6] M. Ludwig, A. Hepp, M. Brunner, and J. Baehr, "CRESS: Framework for Vulnerability Assessment of Attack Scenarios in Hardware Reverse Engineering," in *2021 IEEE Physical Assurance Inspection Electron.s (PAINE)*, 2021.

[7] A. Hepp and G. Sigl, "Tapeout of a RISC-V crypto chip with hardware trojans: a case-study on trojan design and pre-silicon detectability," in *Proc. 18th ACM Int. Conf. Comput. Frontiers*, 2021.

[8] S. Xu. "Wrong exception tval for instruction fetch fault if instruction is crossing page boundary #470." (2020), https : / / github . com / openhwgroup/cva6/issues/470.

[9] B. Lippmann *et al.*, "Integrated flow for reverse engineering of nanoscale technologies," in *Proc. 24th Asia South Pacific Des. Automat. Conf.*, ser. ASPDAC '19, 2019.

[10] K. S. McElvain, *Methods and apparatuses for automatic extraction of finite state machines*, US Patent No. US 6,182,268 B1, Filed Jan. 5th., 1998, Issued Jan. 30th., 2001, Jan. 2001.

[11] P. Subramanyan *et al.*, "Reverse Engineering Digital Circuits Using Structural and Functional Analyses," in *Emerg. Topics Comput., IEEE Trans.*, 2014.

[12] T. Meade, K. Shamsi, T. Le, J. Di, S. Zhang, and Y. Jin, "The Old Frontier of Reverse Engineering: Netlist Partitioning," *J Hardw Syst Secur*, 2018.

[13] M. Werner, B. Lippmann, J. Baehr, and H. Gräb, "Reverse Engineering of Cryptographic Cores by Structural Interpretation Through Graph Analysis," in *2018 IEEE 3rd Int. Verification Secur. Workshop (IVSW)*, 2018.

[14] B. Cakir and S. Malik, "Revealing Cluster Hierarchy in Gate-level ICs Using Block Diagrams and Cluster Estimates of Circuit Embeddings," *ACM Trans. Des. Autom. Electron. Syst.*, 2019.

[15] L. Alrahis *et al.*, "GNN-RE: Graph Neural Networks for Reverse Engineering of Gate-Level Netlists," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 2021.

[16] J. Baehr, A. Bernardini, G. Sigl, and U. Schlichtmann, "Machine Learning and Structural Characteristics for Reverse Engineering," *Integr. VLSI J.*, 2020.

[17] T. Meade, Y. Jin, M. Tehranipoor, and S. Zhang, "Gate-level netlist reverse engineering for hardware security: Control logic register identification," in *2016 IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2016.

[18] T. Meade, S. Zhang, and Y. Jin, "Netlist reverse engineering for high-level functionality reconstruction," in *2016 21st Asia South Pacific Des. Automat. Conf. (ASP-DAC)*, 2016.

[19] M. Brunner, J. Baehr, and G. Sigl, "Improving on State Register Identification in Sequential Hardware Reverse Engineering," in *2019 IEEE Int. Symp. Hardware Oriented Secur. Trust (HOST)*, 2019.

[20] L. Azriel, J. Speith, N. Albartus, R. Ginosar, A. Mendelson, and C. Paar, "A survey of algorithmic methods in IC reverse engineering," *J. Cryptographic Eng.*, 2021.

[21] K. Shamsi, M. Li, K. Plaks, S. Fazzari, D. Z. Pan, and Y. Jin, "IP Protection and Supply Chain Security Through Logic Obfuscation: A Systematic Overview," *ACM Trans. Des. Autom. Electron. Syst.*, 2019.

[22] Y. Yang, Z. Chen, Y. Liu, T.-Y. Ho, Y. Jin, and P. Zhou, "How Secure Is Split Manufacturing in Preventing Hardware Trojan?" In *ACM Trans. Des. Automat. Electron. Syst.*, 2020.

[23] J. Rajendran, O. Sinanoglu, and R. Karri, "Regaining trust in VLSI design: Design-for-trust techniques," *Proc. IEEE*, 2014.

[24] A. Alaql and S. Bhunia, "Scalable Attack-Resistant Obfuscation of Logic Circuits," *arXiv preprint arXiv:2010.15329*, 2020.

[25] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending piracy of integrated circuits," *Comput.*, 2010.

[26] R. Karmakar, S. Chatopadhyay, and R. Kapur, "Encrypt Flip-Flop: A Novel Logic Encryption Technique For Sequential Circuits," *arXiv preprint arXiv:1801.04961*, 2018.

[27] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *2015 IEEE Int. Symp. Hardware Oriented Secur. Trust (HOST)*, 2015.

[28] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," in *2017 IEEE Int. Symp. Hardware Oriented Secur. Trust (HOST)*, IEEE, 2017.

[29] lowRISC. "Ariane Core Github." (2019), https://github.com/lowRISC/ariane.

[30] S. Roshanisefat, H. Mardani Kamali, H. Homayoun, and A. Sasan, "RANE: An Open-Source Formal De-Obfuscation Attack for Reverse Engineering of Logic Encrypted Circuits," in *Proc. 2021 Great Lakes Symp. VLSI*. 2021.

[31] D. Šišejković, F. Merchant, R. Leupers, G. Ascheid, and S. Kegreiss, "Inter-Lock: Logic Encryption for Processor Cores Beyond Module Boundaries," in *2019 IEEE European Test Symp. (ETS)*, 2019.

[32] S. Bhasin and F. Regazzoni, "A survey on hardware trojan detection techniques," in *2015 IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2015.

[33] C. Nigh and A. Orailoglu, "AdaTrust: Combinational Hardware Trojan Detection Through Adaptive Test Pattern Construction," *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, 2021.

[34] B. Lippmann *et al.*, "Physical and Functional Reverse Engineering Challenges for Advanced Semiconductor Solutions," in *2022 Des., Automat. Test Europe Conf. Exhibition (DATE)*, 2022.

[35] M. E. Massad, J. Zhang, S. Garg, and M. V. Tripunitara, "Logic Locking for Secure Outsourced Chip Fabrication: A New Attack and Provably Secure Defense Mechanism," *arXiv:1703.10187 [cs]*, 2017.